Parallelism

Classes of parallelism

Parallelism at multiple levels is now the driving force of computer design across all classes of computers, with energy and cost being the primary constraints.

There are basically two kinds of parallelism in applications:

1. **Data-Level Parallelism** (DLP) arises because there are many data items that can be operated on at the same time
2. **Task-Level Parallelism** (TLP) arises because tasks of work are created that can operate independently and largely in parallel

Computer hardware, in turn, can exploit these two kinds of application parallelism in four major ways:

1. Instruction-Level Parallelism exploits data-level parallelism at modest levels with compiler help using ideas like pipelining and at medium levels using ideas like speculative execution
2. **Vector Architectures** and **Graphics Processor Units** (GPUs) exploit data-level parallelism by applying a single instruction to a collection of data in parallel
3. **Thread-Level Parallelism** exploits either data-level or task-level parallelism in a tightly coupled hardware model that allows for interaction among parallel threads
4. **Request-Level Parallelism** exploits parallelism among largely decoupled tasks specified by the programmer or the operating system

Parallel architectures

There are four ways for hardware to support the data-level and task-level parallelism that go back fifty years: when Michael Flynn studied the parallel computing efforts in the 1960s, he found a simple classification, now called Flynn's Taxonomy, that we still use today.

The four categories he came up with are:

Category name	Acronym	Description
Single instruction	SISD	This category is the uniprocessor. The programmer thinks of it as the standard, sequential computer, but it can exploit ILP.

Category name	Acronym	Description
stream, single data stream		
Single instruction stream, multiple data streams	SIMD	The same instruction is executed by multiple processors using different data streams. SIMD computers exploit data-level parallelism by applying the same operations to multiple data items in parallel. Each processor has its own memory, but there is a single instruction memory and control processor. This architecture is primarily exploited by vector architectures and GPUs.
Multiple instruction streams, single data stream	MISD	Systolic arrays are an example of this architecture.
Multiple instruction streams, multiple data streams	MIMD	Each processor fetches its own instructions and operates on its own data, and it targets task-level parallelism. In general, MIMD is more flexible than SIMD and thus more generally applicable, but is inherently more expensive than SIMD.